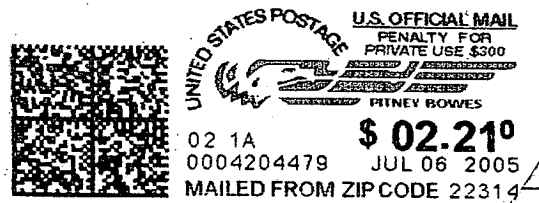
IC2100 RANDOLPH

[redacted]

P.O. BOX 1450
ALEXANDRIA, VA 22313-1450
IF UNDELIVERABLE RETURN IN TEN DAYS

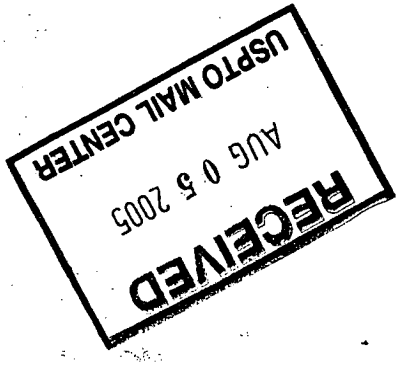
OFFICIAL BUSINESS



AN EQUAL OPPORTUNITY EMPLOYER

FORWARDING ORDER
EXPIRED
MAIL IS FORWARDED FOR
12 MONTHS ONLY
ROUTE # INITIALS

[redacted]



Best Available Copy

IFW

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/844,066	04/26/2001	Michael D. Doyle	021117-000200US	9586

7590 07/06/2005

Edward J. Radlo
Fenwick & West LLP
Two Palo Alto Square
Palo Alto, CA 94306

EXAMINER	
GYORFI, THOMAS A	

ART UNIT	PAPER NUMBER
2135	

DATE MAILED: 07/06/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/844,066	DOYLE ET AL.	
	Examiner	Art Unit	
	Tom Gyorfi	2135	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *04 April 2005*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claim 1 remains for examination. The correspondence filed 4/4/05 did not add, amend, or cancel any claims.

Response to Arguments

2. Applicant's arguments, see the correspondence filed 4/4/05, with respect to the rejection of claim 1 have been fully considered and are persuasive. The rejection of claim 1 under 35 USC 103 has been withdrawn.

Double Patenting

3. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. See *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970);and, *In re Thorington,* 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent is shown to be commonly owned with this application. See 37 CFR 1.130(b).

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

4. Claim 1 is provisionally rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claim 1 of copending Application No. 09/844790. Although the conflicting claims are not identical, they are not patentably distinct from each other because the copending application's disclosure

supports all the limitations of claim 1 of the instant Application. Claim 1 of Application 09/844790 recites verbatim all the limitations of claim 1 of the instant Application with the exception of "configuring a second server to request a cross-certification for a second interval so that the first server is effectively requested to provide independent proof of the existence of the interval and its public key at a point in time witnessed by the first server." However, the disclosure of 09/844790 suggests that this limitation could be added (page 6, lines 12-19 and Figure 13). It would have been obvious to one of ordinary skill in the art at the time the invention was made to add this limitation, as it would render attacks against a single server fruitless (Ibid).

This is a provisional obviousness-type double patenting rejection because the conflicting claims have not in fact been patented.

Conclusion

5. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

- U.S. Patent 4,309,569 issued to Merkle, Ralph C.
- U.S. Patent 5,022,080 issued to Durst et al.
- U.S. Patents 5,136,643, 5,373,561, and 5,781,629 issued to Haber et al.
- U.S. Patents 5,136,646 and 5,422,953 issued to Fischer, Addison M.
- H. Massias and J. Quisquater. *Time and cryptography*, 1997. Université catholique de Louvain, March 1997. TIMESEC Technical Report WP1.

- Stuart Haber and W.-Scott Stornetta. How to Time-Stamp a Digital Document. Journal of Cryptology, 3(2):99--111, 1991
- Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In Sequences'91: Methods in Communication, Security, and Computer Science, pages 329--334. SpringerVerlag, 1992.
- Shamir, A.: RSA for paranoids. CryptoBytes 1 (1995) 1--4. http://citeseer.ist.psu.edu/shamir95rsa.html (pages 14-15)

6. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tom Gyorfi whose telephone number is (571) 272-3849. The examiner can normally be reached on 8:00am - 4:30pm Monday - Friday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kim Vu can be reached on (571) 272-3859. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

TAG
6/23/05

KIM VU
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Notice of References Cited	Application/Control No. 09/844,066	Applicant(s)/Patent Under Reexamination DOYLE ET AL.	
	Examiner Tom Gyorfi	Art Unit 2135	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-5,781,629	07-1998	Haber et al.	713/177
	B	US-5,422,953	06-1995	Fischer, Addison M.	713/172
	C	US-5,373,561	12-1994	Haber et al.	713/157
	D	US-5,136,646	08-1992	Haber et al.	713/178
	E	US-5,136,643	08-1992	Fischer, Addison M.	713/178
	F	US-5,022,080	06-1991	Durst et al.	713/178
	G	US-4,309,569	01-1982	Merkle, Ralph C.	713/177
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	H. Massias and J. Quisquater. Time and cryptography, 1997. Universite catholique de Louvain, March 1997. TIMESEC Technical Report WP1.
	V	Shamir, A.: RSA for paranoids. CryptoBytes 1 (1995) 1–4. http://citeseer.ist.psu.edu/shamir95rsa.html (pages 14-15)
	W	Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In Sequences'91: Methods in Communication, Security, and Computer Science, pages 329–334. SpringerVerlag, 1992.
	X	Stuart Haber and W.-Scott Stornetta. How to Time-Stamp a Digital Document. Journal of Cryptology, 3(2):99–111, 1991

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

UCL
Université
catholique
de Louvain

TIMESEC

Digital Timestamping and the Evaluation of Security Primitives

Time and cryptography

H. Massias
massias@dice.ucl.ac.be

J.-J. Quisquater
jjq@dice.ucl.ac.be

March 1997

WP1, Technical Report

Contents

Overview

The interaction between time and cryptography is a new subject. Nevertheless, the time in cryptography is of high importance especially for electronic contracts.

First we must define what we call *time* in cryptography and the different situations we will deal with. Two of the needs are to timestamp a document to answer the question "When has this document been done?" and to send information in the future.

Some solutions were given in the past but they leave some open problems and none have been standardized. There are two kinds of solutions, the one which need a T.T.P. (Trust Third Party) and the others. A solution to date safely is to link the requests. We will see two solutions based on this idea.

Several protocols have been patented, we will study these US-patents. Then we will see applications of the solutions we commented.

None of the protocols which are described in part (1, 3, 4.1) are new. Nevertheless we have precisely describe the verification protocols and we have made the cryptanalyze of all the algorithms. We have studied in details all the aspect of the protocols and then drawn conclusions.

To finish we will proposed a terminology and an informal method based on this work.

Introduction

We will study two needs of dealing with time in cryptography. First the timestamp and then the "timed-release crypto".

Definition 1 *The* **time-stamp** *of a document is something added to it which proves that the document have been issued* **before, after** *or* **at** *a certain time.*

Timestamping applies to two main problems in cryptography. The first one is to be able to date securely a digital document and the second one is to extend the lifetime of signatures.

Definition 2 *The goal of* **timed-release crypto** *is to encrypt a message so that it can not be decrypted by anyone, until a pre-determined amount of time has passed.*

It consist on "sending information into the future."

1 Timestamping

The reference for timestamping today is the work of Stuart Haber and W.Scott Stornetta ([1]).

1.1 Several situations

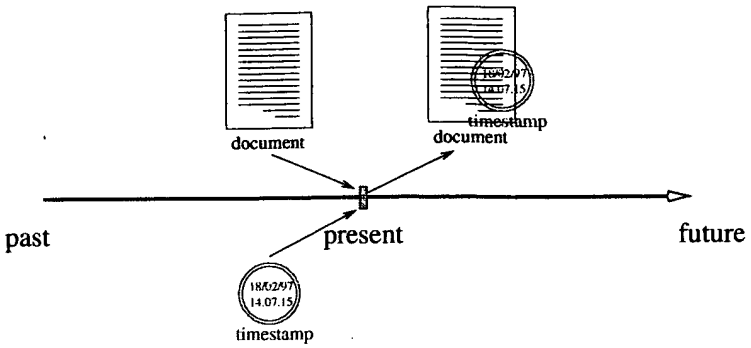
There are several situations in which a document can be timestamped (Figure 1).

In the first situation we make the assumption that we timestamp documents that have just been issued, with the actual time and date.

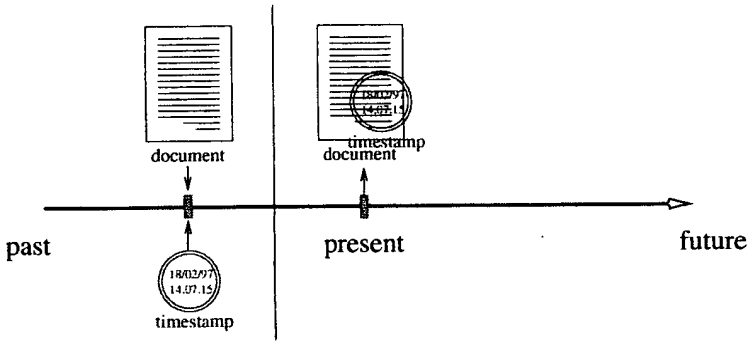
In the second situation, we timestamp today a document that has been issued in the past, with the time when it was made.

We will only consider the first situation. So no matter when the document was issued, we only worry about when it is submitted to be timestamped.

In this situation, it's **easy** to prove that a document was made **after** a given time, but **difficult** to prove that it was made **before** a given time.
To prove that a document was made after a given time, a solution is to include in the timestamp a widely witnessed event that everybody can date.



Situation 1



Situation 2

Figure 1: The different situations

1.2 Several solutions

There are two families of protocols.

- The distributed protocols which work without TTP.

- The linking protocols which require a TTP or only a third party.

In all cases, we will date a hash of the document to be timestamped in order to keep the confidentiality of the original document without losing security.

1.3 Distributed protocol

The protocol Alice wants to date a document y, here's the protocol she uses ([1], [3], [5], [7]):

1. First, by using a pseudo-random generator and by initializing it with y, she computes k values $V_1, V_2, \cdots, V_k$.
2. She interprets this k values as identification of k persons, and sends them y.
3. Each of these persons adds date and time to the document before signing it and sending it back to Alice.
4. Alice keeps these k signatures as timestamp of her document.

Explanations To guarantee the safety of this protocol, we have to assume that Alice cannot corrupt k people, so we must choose k big enough.

Initializing the generator with the document y enables the people who wants to check the timestamp to be sure that Alice didn't choose the identity of the k people. We can initialize the pseudo-random generator with a random number that we add in the timestamp (for the verification) but it didn't bring something interesting. We can also imagine that Alice could choose this number in order to obtain identities of people that she can corrupt. A similar idea, but with the protocol we have just defined, is that Alice could make minor changes to his document in order to "choose" the k numbers generated. This must be study for an implementation because it's linked to k, to the hash function used by Alice and to the pseudo-random generator.

The first major drawback of this protocol is the need of enough people able to answer immediately to Alice's request.

The second drawback is the *lifetime of the signatures* which are send back to Alice. This problem will appear each time we use signatures. For this reason Burt Kalisky says that a timestamping protocol must not rely on any secret information.

1.4 Linking protocol

We will denote the TTP by TSS (Time Stamping Service).

1.4.1 Method using chain

A natural solution when we take advantage of a TTP is the following.

We suppose that Alice wants to timestamp y.

1. Alice sends y to the TSS.
2. The TSS adds the date and time, signs and sends it back to Alice.

If the TSS is honest, the only problem is once again the lifetime of the signature.

Let's now improve this protocol in order to reduce the risk of collusion between Alice and the TSS.

To do this a solution consists on linking Alice's document with the documents that the TSS signed before and after, to built an unmodifiable chronological chain. Like this we can be sure that Alice's document was dated before the one which follows and after the one which precedes.

Protocol: ([1]) Let S_k be the TSS signing function, and H a hash function. We suppose that Alice wants to timestamp y_n. It's the n^{th} request made to the TSS.

1. Alice sends y_n and ID_n (her identity) to the TSS.
2. The TSS sends back to Alice:

$$s = S_k(n, t_n, ID_n, y_n; L_n)$$

where t_n is the date and time, and

$$L_n = (t_{n-1}, ID_{n-1}, y_{n-1}, H(L_{n-1}))$$

3. When the next request will be issued, the TSS will send ID_{n+1} to Alice.

(s, ID_{n+1}) is the timestamp for y_n.

L_n is the linking information.

Verification

- First you check $s = S_k(n, t_n, ID_n, y_n; L_n)$.
- Then you ask ID_{n+1} for his timestamp $(n+1, t_{n+1}, ID_{n+1}, y_{n+1}; L_{n+1})$, $L_{n+1} = (t_n, ID_n, y_n, H(L_n))$ and you check if it works.
- You ask ID_{n-1} for his timestamp and see one more time if it works.

Now you can do the same with ID_{n+2}, ID_{n-2} and so on.

Reliability

If we assume that the signature scheme used by the TSS is reliable and that his secret key is still secret, attacks against this scheme can only be made with the collaboration of the TSS.

Let's see the different attacks and what they require.

- We are unable to insert a timestamp in the chain because each message is numbered.
- To make a false timestamp we can replace a document in the chain. To do that we must collude with the TSS and with the people whose requests precedes and follows the one we want to replace. Then we must find a collision in the hash function.

  Let's see in details what it is like.

  We suppose that we want to replace the timestamp corresponding to the n^{th} request. We assume that we want to change the identity and the hashing, i.e. we want to replace $(n, t_n, ID_n, y_n; L_n)$ with $(n, t_n, ID'_n, y'_n; L_n)$.

  - We must have the collusion of ID_{n-1} to replace (s, ID_n) with (s, ID'_n).
  - Then we must have the collusion of ID_{n+1} too. He must replace $L_{n+1} = (t_n, ID_n, y_n, H(L_n))$ with $L'_{n+1} = (t_n, ID'_n, y'_n, H(L_n))$ determined in such a way that $H(L'_{n+1}) = H(L_{n+1})$ in order to be coherent with the timestamp of the $(n+2)^{\text{th}}$ demand.

  If $ID'_n = ID_n$ then the first condition is no more necessary.

  If H is a "good" hash function then the last condition is unworkable.

- Another method to brake the scheme with the collusion of the TSS is to construct a false chain of timestamps long enough to exhaust the most suspicious challenger. To guard against that we can imagine that the TSS publish $H(L_m)$ every day if m is his last request, in order to prevent the modification of $H(L_m)$).

Remarks

A drawback is that everyone must keep all his timestamps to enable the check of the other timestamps. A solution to this is to link the document not only with the preceding and the following one but with the k's.

1. The protocol is the same than the last one except that the linking information becomes

$$L_n = [(t_{n-k}, ID_{n-k}, y_{n-k}, H(L_{n-k})), \cdots,$$

$$(t_{n-1}, ID_{n-1}, y_{n-1}, H(L_{n-1}))].$$

2. When the k next requests have been proceeded, the TSS sends $(ID_{n+1}, \cdots, ID_{n+k})$ to Alice.

A new drawback is that the size of the timestamp grows up, but in compensation it increases the security. To make the same attack that we described before we must not only corrupt ID_{n-1} and ID_{n+1} but $ID_{n-k}, \cdots, ID_{n-1}, ID_{n+1}, \cdots, ID_{n+k}$ too.

More explanation

- Why do we impose to hash the linking information?

  Let's assume that we include L_{n-1} instead of $H(L_{n-1})$ in L_n. Then we include L_{n-2} because it's in L_{n-1} and so on. After each new request the timestamps's size increase, so that becomes quickly unworkable.

  If we decide to link only with the last request $L_n = (t_{n-1}, ID_{n-1}, y_{n-1})$ in order to avoid the use of a hash function, the attack we described earlier is much simple. Now it only requires to be able to modify two requests spaced out one apart. It's no more necessary to find a collision in the hash function.

- In the information L_n that we add in the timestamp, $H(L_{n-1})$ is useful to make the timestamp number n dependent on the preceding ones and to keep a constant size at the same time. $t_{n-1}, ID_{n-1}, y_{n-1}$ are useful for the checking.

1.4.2 Method using tree structure

This scheme uses hash function but **no** keys, i.e. no secret information.

An idea of the protocol is briefly given in ([2]) and an implementation of it is describe by Surety Technologies (see paragraph 4.2.2). We have use that to deduce the protocol which follows.

Protocol (Figure 2)

Let H be a hash function.

We suppose that the TSS receives n requests in the same time unit. The TSS calculates the value HV_i by hashing the documents via a tree structure. The last value HV_i is widely published so that nobody can change it, but everybody can easily get it.

In Figure 2 the y_X, $X = A, B, \cdots, H$ are the documents that the users want to timestamp.

Alice's timestamp contains the information necessary to rebuild the tree branch his document belongs to. The timestamp for y_B is (y_A, right), (H_2, left), (H_6, left).

Verification

The checking simply consists on rebuilding the tree branch and comparing the value that we obtain with the one published for the time unit t_i: HV_i.

Reliability

An attack could be to construct a false tree branch starting with the document we want to timestamp and finishing with the value HV_i corresponding to the time we want to date with.

The assumption that we make is that the tree branch can only be rebuilt with the original values.

We must make sure that this representation doesn't make it easier to find attacks against hash functions.

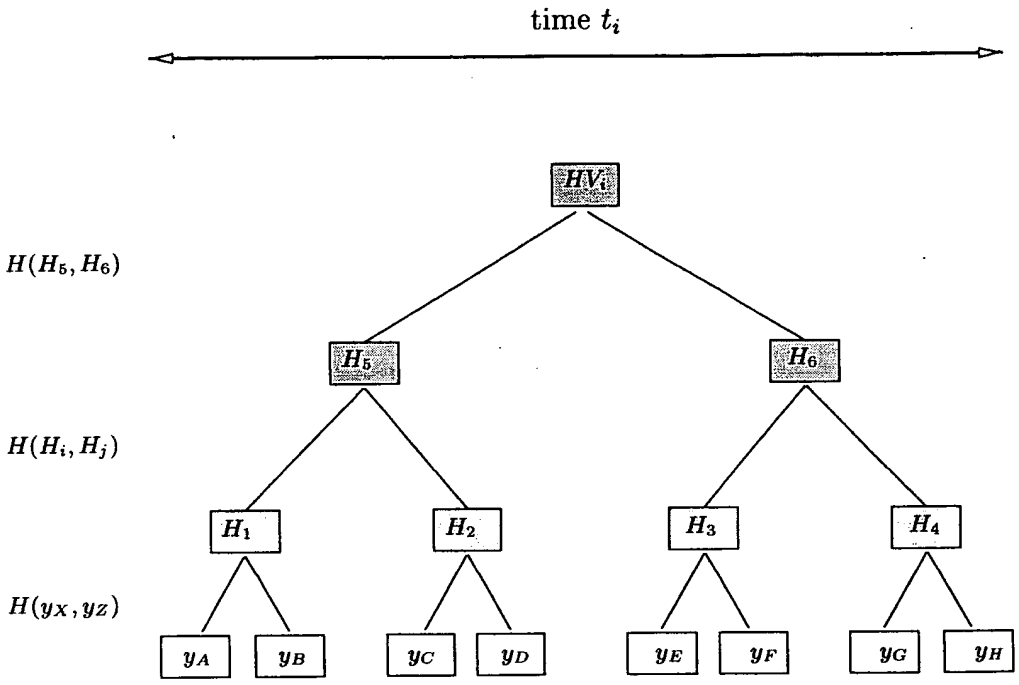


Figure 2: Tree structure

Comments

An advantage is that the checking doesn't depend on the stocking of their timestamps by the other users as opposed to the chaining protocol.

A drawback is that we must have enough requests for each time unit. This must be studied before an implementation.

An improvement

A variation of this protocol consists on linking the values HV_i together with a hash function (Figure 3). We consider the value SHV_{i-1} (Super Hash Value) which is a hashing of all the last HV_j values and we compute $SHV_i = H(SHV_{i-1}, HV_i)$.

An attack

Assume that Charlie wants to make an attack. We suppose that he can't publish the SHV instead of the TSS.

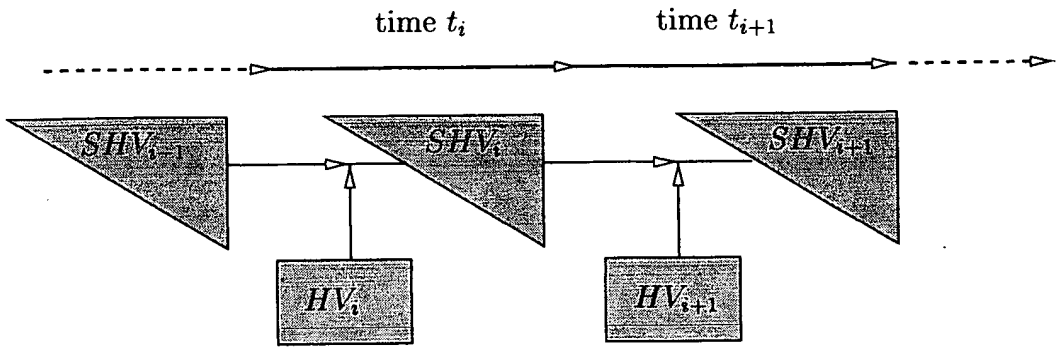


Figure 3: An improvement

Charlie intercepts all the documents to be timestamped and takes the place of the TSS. He will invalidate all the timestamps of the users. If there's no authentication protocol between the TSS and the users it will be very easy and the users will discover deceit only when the SHV will be published. A way to prevent this is that the TSS signs the timestamps. The attack which consist on building a false branch tree will thus become more difficult because it requires the collusion of the TSS.

1.5 The Network Time Protocol: NTP

The introduction of the NTP had enable the computers connected to Internet to know the exact time. The protocol is briefly explain in ([11]). A study of the security has been done by M. Bishop ([12]).

Its only use is to become the exact time, but it didn't enables to timestamp at all.

Explanations and references can be found on the Internet at *http://www.belnet.be/services/ntp.html.*

2 Patents

For the moment four US-patents concerning timestamping protocols have been register. All are from Stuart A. Haber and Wakefield S. Stornetta.

Filed	Date	N°	Title
Aug. 2, 1990	Aug. 4, 1992	5,136,647	Method for secure timestamping of digital documents
Mar. 8, 1991	Aug. 4, 1992	5,136,646	Digital document time-stamping with catenate certificate
Dec. 21, 1992	Dec. 13, 1994	5,373,561	Method of extending the validity of a cryptographic certificate
Nov. 22, 1993	May 30, 1995	RE. 34,954	Method for secure timestamping of digital documents

2.1 US-patent n° 5,136,646

H is a hash function.

1. Alice sends y_n and ID_n (her identity) to the TSS.
2. The TSS sends back to Alice:

$$(n, t_n, ID_n; C_n)$$

where t_n is the date and time, and

$$C_n = H(n, y_n, ID_n, t_n; C_{n-1})$$

$(n, t_n, ID_n; C_n)$ is the timestamp for y_n.

2.2 US-patent n° 5,136,647

Let s be the TSS signing function.

1. Alice sends y_n and ID_n (her identity) to the TSS.
2. The TSS sends back to Alice:

$$s(C_n)$$

where

$$C_n = (n_{|8}, y_{n|8}, ID_{n|8}, t_{n|8}; y_{n-1|8}, ID_{n-1|8}, t_{n-1|8})$$

$n_{|8}$ means that we truncate n to his last 8 bits.

A big mistake here is to use truncation as a hash function, because it's a very weak hash function.

Another drawback is that the document is only linked with the last request and not with all the preceding ones.

In this form, this patent is useless.

2.3 US-patent n° 5,373,561

The method just consists on timestamping the document with its certificate like we will see in (4).

2.4 US-patent n° Re.34,954

This patent is just the patent n° 5,136,647 which has been reissued with some more claims.

3 Timed release crypto

This problem was first discussed by Timothy May ([9]). Some solutions have been given by Rivest, Shamir and Wagner in ([10]).

Let's now see just the principle of what can be done when we take advantage of a TTP. A good and precise protocol is explained in ([10]).

1. Alice encrypts a message $M' = E(M, K)$ with the key K and wants that M' can not be decrypted before a time t_i.
2. She encrypts K with the public key e_{t_i} and publishes the encrypted value.
3. At time t_i the TTP publishes d_{t_i} which is the secret key corresponding to e_{t_i}.

We believe that this feature can easily be added to a timestamping service.

4 Applications

4.1 Current technical solutions

As we said in the introduction, timestamping has two main applications. The first one is to answer the question: *"When was this document issued? "*.

The second one is to extend lifetime of certain sort of certificates when the protocol used is compromised.

Let's take the example of signatures. Today signatures have a lifetime, it's the time before the secret key is compromised. After that all the signatures that have been made with this key are called into question. That's really a big problem. We can also imagine that someone which wants to repudiate a signature he made, can dishonestly report the compromise of his private key. Then all the signatures he made are invalidated. By doing so any user is able to disavow a signature he regrets.

The protocol (see [2], [6])

Let be D the document and s its signature.

We just timestamp (D, s) and then we are as sure of the validity of the signature as we were when it was timestamped.

In the same way, we can renew timestamps by re-timestamping the document and its first timestamp.

4.2 Implementations

We only know three implementations. Two of them use PGP and work nearly in the same way.

4.2.1 PGP Digital Timestamping services

These services are accessible by email, the explanations are available on the web: *http://www.itconsult.co.uk/stamper.htm, http://www.eskimo.com/ weidai/timestamp.html.*

Here's a brief description of the first service.

Each signature made has a serial number.

- The TSS signs the message.

- He stores all the signatures that he makes. All the detached signatures(serial number, time and date) that have been issued can be inspected.
- Every day the TSS generates two files, one showing the last signature serial number made, and the other containing all the detached signatures made on that day.

  A signature of the files is published every week in a newsgroup.

4.2.2 Digital Notary

It's a product of Surety Technologies (*http://www.surety.com*).

This implementation uses the tree structure (1.4.2).

The time unit is one second. A unique serial number is attached to each document.

The protocol

1. The customer has a software on his personal computer and is connected to the TSS via Internet.
2. Using this software he hashes the original document in a 288 bits hash. He has the option to treat this hash digest singly, or combine it with hash digests from other documents (an "aggregated hash") before transmitting to the TSS.
3. The TSS combine this document(s) with the others that arrived at the same second by hashing them via a tree structure.
4. Each second the TSS construct the SHV. It's published in several places accessible via the network and on CD-ROM too. A value is published every week on the Sunday New-York Times.
5. The TSS sends the customer all that he needs to certify his timestamp: the hash of each document, the computation path followed when all hashes and aggregated hashes were mathematically combined, and the time and date of the HV.

5 Proposed terminology and informal method

We will call STA (Secure Time Authority) the agency which will make secure the use of time for digital documents.

For us *timestamping* is the operation which consist on building the proof that a document was submitted to the STA on a given date.
What we call *timed-release crypto* is the action of keeping an information secret before a given time.

In *timestamping* and *timed-release crypto* there are two global concept. The STA can make the global certification (that we call "global service") or it can give the user the material to do the certification ("self service").

5.1 Global service

A global service for timestamping will consist for the STA on timestamping a document and linking it with other to improve security (see 1.4).

The solution we will adopt is to add the document, the time, the date and some linking information which comes from all the documents that have already been timestamped. At last the STA signs the result before sending it to the user.

A global service for *timed release crypto* would be for the STA to keep the document secret until the expiration date. An alternative would be for the STA to encrypt the document and to keep secret both the document and the key or only the key.

5.2 Self service

We could imagine a self service for timestamping as follow. The STA distributes certified time and date to the user who adds this as a timestamp to his document. The problem is that it just proves that the document has been timestamped after the time contained in the timestamp because the user can keep the timestamp and use it after, or can copy it.

We can prevent from copying the timestamp in the same way as in electronic money. To prevent from using the timestamp later we should force the user to prove that he used the timestamp as soon as he got it.
We can find a way to prove that the timestamp has been used but the problem is to date the proof.

If it's an "on-line" protocol, a solution could be to give a lapse of time beginning when the timestamp is given by the STA. After this lapse of time, if the STA has not receive the proof, the timestamp is invalidated.

A self service for *timed release crypto* works on the idea given in 3.

The STA just produces keys: a public one and the corresponding secret one, and propose a protocol. The user uses the protocol with the public key to encrypt his document. He can now publish the encrypted text which will only be understandable when the STA will publish the secret key.

5.3 Our choice

We will use the NTP to check the STA clock.
We propose to use the global service for timestamping and the self service for timed release crypto because they seems to us the most feasible and they answer the best to the requirements of the project.

References

[1] S. Haber and W.-S. Stornetta, *"How to Time-Stamp a Digital Document"*, Journal of Cryptology, v.3,n.2,1991,pp. 99-112.

[2] D. Bayer, S. Haber and W.-S. Stornetta, *"Improving the Efficiency and Reliability of Digital Time-Stamping"*, Sequences'91: Methods in Communication, Security, and Computer Science; Springer Verlag, 1992, pp. 329-334.

[3] B. Schneier *Applied Cryptography (second edition)*, John Wiley & Sons, inc , 1996.

[4] S. Haber and W.-S. Stornetta, *"Digital document time-stamping with catenate certificate"*, US-patent n° 5,136,646; 1992.

[5] S. Haber and W.-S. Stornetta, *"Method for secure timestamping of digital documents"*, US-patent n° 5,136,647; 1992.

[6] S. Haber and W.-S. Stornetta, *"Method of extending the validity of a cryptographic certificate"*, US-patent n° 5,373,561; 1994.

[7] S. Haber and W.-S. Stornetta, *"Method for secure timestamping of digital documents"*, US-patent n° RE. 34,954; 1995.

[8] S. Haber, B. Kalisky,W-S. Stornetta *"How do digital Time-Stamps support Digital signatures?"*, Cryptobytes, autumn 1995.

[9] Timothy C. May *"Timed-release crypt"*, February 1993.

[10] R. Rivest, A. Shamir, D. Wagner *"Time-lock puzzles and timed-release crypto"*, March 10, 1996.

[11] *"Belnet Info"*, Numéro de référence, 1996, pp.13.

[12] M. Bishop *"A security Analysis of the NTP Protocol (DRAFT)"*, January 1990.

VOLUME 1, NUMBER 3 — AUTUMN 1995

RSA LABORATORIES'
CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

RSA for Paranoids

Adi Shamir
Applied Math Department
The Weizmann Institute of Science
Rehovot 76100, Israel

One of the most important decisions in practical implementations of the RSA cryptosystem is the choice of modulus size. It is clear that the standard size of 512 bits no longer provides adequate protection, and should be substantially increased. However, the time complexity of modular exponentiation grows rapidly with the size of the modulus, and thus it is difficult to choose a size which combines efficient operation with long term security. In this note we describe a new variant of the RSA cryptosystem called "unbalanced RSA", which makes it possible to increase the modulus size from 500 bits to 5,000 bits without any speed penalty.

Conventional RSA

The security of the RSA cryptosystem depends (but is not provably equivalent to) the difficulty of factoring the modulus n, which is the product of two equal size primes p and q. Until recently, the minimum recommended size of n was 512 bits. However, recent advances in factoring algorithms make it necessary to increase this size. Since the time complexity of RSA computations is already substantial, and grows cubically with the size of the modulus, the new size should by necessity be a compromise between efficiency and security. The choice is particularly difficult for paranoid organizations whose encrypted messages should remain secret for several decades, since it is almost impossible to predict the progress of factoring algorithms over such a long period of time. The only reasonable course of action is to use huge margins of safety, but this will make the RSA operations extremely slow.

All the known factoring algorithms can be divided into two broad types: algorithms whose running time depends on the size of the factors, and algorithms whose running time depends only on the size of the factored number n. The oldest factoring algorithms typically searched for the smallest factor p of n, and were thus of the first type. However, modern algorithms tend to use indirect approaches which require the same time to find a single digit or a fifty digit prime factor of n.

The fastest factoring algorithm of the first type is currently the elliptic curve method. Its asymptotic running time is $\exp(O((\ln(p))^{1/2} \cdot (\ln\ln(p))^{1/2}))$, but its basic operations are very slow. The largest factor ever found in practice with this algorithm was about 145 bits long (A. Lenstra, private communication), and it is very unlikely that this algorithm will be able to find the 256 bit factors of 512 bit RSA keys in the next few years.

Factoring algorithms of the second type are much faster, since they can use a wider array of mathematical techniques. The best algorithm of this type is currently the general number field sieve. It has an asymptotic complexity of $\exp(O((\ln(n))^{1/3} \cdot (\ln\ln(n))^{2/3}))$, and is believed to be capable of factoring 512 bit moduli in 10,000 to 15,000 MIPS-years (A. Lenstra, private communication). This is

(continued on page 3)

Adi Shamir is professor at the Applied Math Department of the Weizmann Institute of Science, Israel. He is a co-inventor of the RSA cryptosystem and can be contacted at shamir@wisdom.weizmann.ac.il.

How Do Digital Time-Stamps Support Digital Signatures?

Answered by
Stuart Haber, Burt Kaliski and **Scott Stornetta**

Consider two questions that may be asked by a computer user as he or she views a digital document or on-line record. (1) Who is the author of this record — who wrote it, approved it, or consented to it? (2) When was this record created or last modified?

In both cases, the question is one about exactly *this* record—exactly this sequence of bits—whether it was first stored on this computer or was created somewhere else and then copied and saved here. An answer to the first question tells *who & what*: who approved exactly what is in this record? An answer to the second question tells *when & what*: when exactly did the contents of this record first exist?

Both of the above questions have good solutions. A system for answering the first question is called a *digital signature* scheme. Such a system was first proposed in [2] and there is a wide variety of accepted designs for an implementation of this kind of system [4,5].

A system for answering the second question is called a *digital time-stamping scheme*. Such systems were described in [1,3], and an implementation is commercially available from Surety Technologies (*http://www.surety.com/*).

Digital time-stamps increase the longevity of digitally-signed records.

Any system allowing users to answer these questions reliably for all their records must include two different sorts of procedures. First, there must be a *certification* procedure with which (1) the author of a record can "sign" the record, or (2) any user can fix a record in time. The result of this procedure is a small certifying file, a *certificate* if you will, that captures the result of this procedure. Second, there must be a *verification* procedure by which any user can check a record and its accompanying certificate to make sure it correctly answers (1) who and what? or (2) when and what? about the record in question.

The "certificate" returned by the certification procedure of a digital signature system is usually called a *signature*; it is a signature for a particular signer (specifying whom) and for a particular record (specifying what). In order to be able to "sign" documents, a user registers with the system by using special software to compute a pair of numbers called keys, a *public key* and a corresponding *private key*. The private key should only be available to the user to whom it belongs, and is used (by the certification or "signing" procedure) in order to sign documents; it is by employing the user's private key that the signature and the record are tied to that particular user. The public key may be available to many users of the system, and is used by the verification procedure. That is, the verification procedure takes a particular record, a particular user's public key, and a putative signature for that record and that user, and uses this information to check whether the would-be signature was correctly computed using that record and the corresponding private key.

Special computational methods are employed for signing documents and for verifying documents and signatures; when these methods are carefully implemented, they have the remarkable property that the knowledge of a user's public key does not enable an attacker or hacker to figure out the user's corresponding private key. Of course, if, either through carelessness or deliberate intent, someone else—a hacker, for example—gains access to the user's private key, then this person will be able to "forge" the legitimate user's signatures on documents of the hacker's choice. At that point, even the value of legitimately signed records can be called into question.

The "certificate" returned by the certification procedure of a digital time-stamping system is a certificate for a particular record (specifying what) at a particular time (specifying when). The procedure works by mathematically linking the bits of the record to a "summary number" that is widely witnessed by and widely available to members of the public—including, of course, users of the system. The computational methods employed ensure that only the record in question can be linked, according to the "instructions" contained in its time-stamp

Stuart Haber is a research scientist in the Security Research Group at Bellcore, Burt Kaliski is chief scientist at RSA Laboratories, and W. Scott Stornetta is Chairman of Surety Technologies. Surety Technologies was founded by Stornetta and Haber in 1994 as a spin-off from Bellcore, with a mandate to commercialize the digital time-stamping technology developed by Bellcore. The authors can be contacted at stuart@bellcore.com, burt@rsa.com and scotts@surety.com.

certificate, to this widely witnessed summary number; this is how the particular record is tied to a particular moment in time. The verification procedure takes a particular record and a putative time-stamp certificate for that record and a particular time, and uses this information to validate whether that record was indeed certified at the time claimed by checking it against the widely available summary number for that moment.

Two features of a digital time-stamping system are particularly helpful in enhancing the integrity of a digital signature system. First, a time-stamping system cannot be compromised by the disclosure of a key. This is because digital time-stamping systems do not rely on keys, or any other secret information, for that matter. Second, following the technique introduced in [1], digital time-stamp certificates can be *renewed* so as to remain valid indefinitely.

With these features in mind, consider the following situations.

It sometimes happens that the connection between a person and his or her public signature key must be revoked—for example, if the user's secure access to the private key is accidently compromised; or when the key belongs to a job or role in an organization that the person no longer holds. Therefore the person-key connection must have time limits, and the signature verification procedure should check that the record was signed at a time when the signer's public key was indeed in effect. And thus when a user signs a record that may be checked some time later—perhaps after the user's key is no longer in effect—the combination of the record *and* its signature should be certified with a secure digital time-stamping service.

There is another situation in which a user's public key may be revoked. Consider the case of the signer of a particularly important document who later wishes to repudiate his signature. By dishonestly reporting the compromise of his private key, so that all his signatures are called into question, the user is able to disavow the signature he regrets. However, if the document in question was digitally time-stamped together with its signature (and key-revocation reports are time-stamped as well), then the signature cannot easily be disavowed in this way. This is the recommended procedure, therefore, in order to preserve the *non-repudiability* desired of digital signatures for important documents.

The statement that private keys cannot be derived from public keys is an over-simplification of a more complicated situation. In fact, this claim depends on the computational difficulty of certain mathematical problems. As the state of the art advances—both the current state of algorithmic knowledge, as well as the computational speed and memory available in currently available computers—the maintainers of a digital signature system will have to make sure that signers use longer and longer keys. But what is to become of documents that were signed using key lengths that are no longer considered secure? If the signed document is digitally time-stamped, then its integrity can be maintained even after a particular key-length is no longer considered secure.

Time-stamping strengthens non-repudiation of digital signatures.

Of course, digital time-stamp certificates also depend for their security on the difficulty of certain computational tasks concerned with so-called one-way hash functions. (All practical digital-signature systems depend on these functions as well.) Those who maintain a secure digital time-stamping service will have to remain abreast of the state of the art in building and in attacking one-way hash functions. Over time, they will need to upgrade their implementation of these functions, as part of the process of renewal [1]. This will allow time-stamp certificates to remain valid indefinitely.

References

[1] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In R.M. Capocelli, A. De Santis, U. Vaccaro, editors, *Sequences II: Methods in Communication, Security, and Computer Science*, pp. 329-334, Springer-Verlag, New York (1993).

[2] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22: 644-654, 1976.

[3] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, Vol. 3, No. 2, pp. 99-111 (1991).

[4] National Institute of Standards and Technology (NIST). FIPS Publication 186: Digital Signature Standard, May 19, 1994.

[5] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120-126, February 1978.

Improving the Efficiency and Reliability of Digital Time-Stamping*

Dave Bayer†
Barnard College
Columbia University
New York, N.Y. 10027 U.S.A.
dab@math.columbia.edu

Stuart Haber
Bellcore
445 South Street
Morristown, N.J. 07960 U.S.A.
stuart@bellcore.com

W. Scott Stornetta
Bellcore
445 South Street
Morristown, N.J. 07960 U.S.A.
stornetta@bellcore.com

March 1992

Abstract

To establish that a document was created after a given moment in time, it is necessary to report events that could not have been predicted before they happened. To establish that a document was created before a given moment in time, it is necessary to cause an event based on the document, which can be observed by others. Cryptographic hash functions can be used both to report events succinctly, and to cause events based on documents without revealing their contents. Haber and Stornetta have proposed two schemes for digital time-stamping which rely on these principles [HaSt 91].

We reexamine one of those protocols, addressing the resource constraint required for storage and verification of time-stamp certificates. By using trees, we show how to achieve an exponential increase in the publicity obtained for each time-stamping event, while reducing the storage and the computation required in order to validate a given certificate.

We show how time-stamping can be used in certain circumstances to extend the useful lifetime of different kinds of cryptographic certifications of authenticity, in the event that the certifying protocol is compromised. This can be applied to digital signatures, or to time-stamping itself, making the digital time-stamping process renewable.

*Appeared in *Sequences II: Methods in Communication, Security, and Computer Science*, eds. R. Capocelli, A. De Santis, and U. Vaccaro, pp. 329–334, Springer-Verlag (New York, 1993).

†Partially supported by NSF grant DMS-90-06116.

1 Introduction

Causality fixes events in time. If an event was determined by certain earlier events, and determines certain subsequent events, then the event is sandwiched securely into its place in history. Fundamentally, this is why paper documents have forensic qualities allowing them to be accurately dated and examined for signs of after-the-fact tampering. However, documents kept in digital form need not be closely tied to any physical medium, and tampering may not leave any tell-tale signs in the medium.

Could an analogous notion of causality be applied to digital documents to correctly date them, and to make undetected tampering infeasible? Any solution would have to time-stamp the data itself, without any reliance on the properties of a physical medium, and would be especially useful and trustworthy if the date and time of the time-stamp could not be forged.

In [HaSt 91], Haber and Stornetta posed this problem, and proposed two solutions. Both involve the use of cryptographic hash functions (discussed in §2 below), whose outputs are processed in lieu of the actual documents. In the *linking* solution, the hash values of documents submitted to a time-stamping service are chained together in a linear list into which nothing can feasibly be inserted or substituted and from which nothing can feasibly be deleted. This latter property is insured by a further use of cryptographic hashing. In the *random-witness* solution, several members of the client pool must date and sign the hash value; their signatures form a composite certification that the time-stamp request was witnessed. These members are chosen by means of a pseudorandom generator that uses the hash of the document itself as a seed. This makes it infeasible to deliberately choose which clients should and should not act as witnesses.

In both of these solutions, the record-keeping requirements per time-stamping request are proportional to the number of (implicit) observers of the event. In §3 below we address the following problem: What if an immense flood of banal transactions want their time-stamps to become part of the historical record, but history just isn't interested? We propose to merge many unnoteworthy time-stamping events into one noteworthy event, using a tournament run by its participants. The winner can be easily and widely publicized. Each player, by remembering a short list of opponents, can establish participation in the tournament. We do this by building trees in place of the linked list of the linking solution, thus achieving an exponential increase in the number of observers. Such hash trees were previously used by Merkle [Merk 80] for a different purpose, to produce authentication certificates for a directory of public enciphering keys.

There are several ways in which a cryptographic system can be compromised. For example, users' private keys may be revealed; imprudent choice of key-lengths may be overtaken by an increase in computing power; and improved algorithmic techniques may render feasible the heretofore intractable computational problem on which the system is based. In §4 below we show how time-stamping can be used in certain circumstances to extend the useful lifetime of digital signatures. Applying the same technique to time-stamping itself, we demonstrate that digital time-stamps can be

renewed.

Finally, in §5 we discuss the relationships between the different methods of digital time-stamping that have been proposed.

2 Hash functions

The principal tool we use in specifying digital time-stamping schemes, here as in [HaSt 91], is the idea of a cryptographic hash function. This is a function compressing digital documents of arbitrary length to bit-strings of a fixed length, for which it is computationally infeasible to find two different documents that are mapped by the function to the same *hash value*. (Such a pair is called a *collision* for the hash function.) Hence it is infeasible to fabricate a document with a given hash value. In particular, a fragment of a document cannot be extended to a complete document with a given hash value, unless the fragment was known before the hash value was created. In brief, a hash value must follow its associated document in time.

There are practical implementations of hash functions, for example those of Rivest [Riv 90] and of Brachtl, *et al.* [BC$^+$ 88], which seem to be reasonably secure.

In a more theoretical vein, Damgård defined a family of *collision-free hash functions* to be a family of functions $h : \{0,1\}^* \to \{0,1\}^l$ compressing bit-strings of arbitrary length to bit-strings of a fixed length l, with the following properties:

1. The functions h are easy to compute, and it is easy to pick a member of the family at random.

2. It is computationally infeasible, given a random choice of one of these functions h, to find a pair of distinct strings x, x' satisfying $h(x) = h(x')$.

He gave a constructive proof of their existence, on the assumption that there exist one-way "claw-free" permutations [Dam 87]. For further discussion of theoretical questions relating to the existence of families of cryptographic hash functions (variously defined) see [HaSt 91] and the references contained therein.

In the rest of this paper, we will assume that a cryptographic hash function h is given: either a particular practical implementation, or one that has been chosen at random from a collision-free family.

3 Trees

In the linking scheme, the challenger of a time-stamp is satisfied by following the linked chain from the document in question to a time-stamp certificate that the challenger considers trustworthy. If a trustworthy certificate occurs about every N documents, say, then the verification process may require as many as N steps. We may reduce this cost from N to $\log N$, as follows.

Suppose we combine the hash values of two users' documents into one new hash value, and publicize only the combined hash value. (We will consider a "publicized"

value to be trustworthy.) Either participant, by saving his or her own document as well as the other contributing hash value, can later establish that the document existed before the time when the combined hash value was publicized.

More generally, suppose that N hash values are combined into one via a binary tree, and the resulting single hash value is widely publicized. To later establish priority, a participant need only record his own document, as well as the $\lceil \log_2 N \rceil$ hash values that were directly combined with the document's hash value along the path to the root of the tree. In addition, along with each combining hash value, the user needs to record its "handedness," indicating whether the newly computed value was placed before or after the combining hash value. Verification consists simply of recomputing the root of the tree from this data.

Once hashing functions are chosen, such a scheme could be carried out like a world championship tournament: Heterogeneous local networks could govern local subtrees under the scrutiny of local participants, and regional "winners" could be combined into global winners under the scrutiny of all interested parties. Global communication facilities are required, and a broadcast protocol must be agreed upon, but no centralized service bureau need administer or profit from this system. For example, given any protocol acceptable separately to the western and eastern hemispheres for establishing winners for a given one-hour time period, the winners can be broadcast by various members of the respective hemispheres, and anyone who wants to can carry out the computations to determine the unique global winner for that time period. Winners for shorter time periods can similarly be combined into unique winners for longer time periods, by any interested party.

At a minimum, daily global winners could be recorded in newspaper advertisements, to end up indefinitely on library microfilm. The newspaper functions as a widely available public record whose long-term preservation at many locations makes tampering very difficult. An individual who retains the set of values tracing the path between his document and the hash value appearing in the newspaper could establish the time of his document, without any reliance on other records. Anyone who wishes to be able to resolve time-stamps to greater accuracy needs only to record time-stamp broadcasts to greater accuracy.

4 Using time-stamping to extend the lifetime of a threatened cryptographic operation

The valid lifetime of a digitally signed document can be extended with digital time-stamping, in the following way. Imagine an implementation of a particular digital signature scheme, with a particular choice of key lengths, and consider a plaintext document D and its digital signature σ by a particular user. Now let the pair (D, σ) be time-stamped. Some time later the signature may become invalid, for any of a variety of reasons, including the compromise of the user's private key, an increase in available computing power making signatures with keys of that length unsafe, or the discovery of a basic flaw in the signature scheme. At that point, the document-signature pair becomes questionable, because it may be possible for someone other

than the original signer to create valid signatures.

However, if the pair (D, σ) was time-stamped at a time before the signature was compromised, then the pair still constitutes a valid signature. This is because it is known to have been created at a time when only legitimate users could have produced it. Its validity is not in question even though new signatures generated by the compromised method might no longer be trustworthy.

The same technique applies to other instances of cryptographic protocols. In particular, the technique can be used to renew the time-stamping process itself. Once again, imagine an implementation of a particular time-stamping scheme, and consider the pair (D, C), where C is a valid time-stamp certificate (in this implementation) for the document D. If (D, C) is time-stamped by an improved time-stamping method before the original method is compromised, then one has evidence not only that the document existed prior to the time of the new time-stamp, but that it existed at the time stated in the original certificate. Prior to the compromise of the old implementation, the only way to create a certificate was by legitimate means. (The ability to renew time-stamps was mentioned in [HaSt 91] but an incorrect method was given. The mistake of the previous work was in assuming that it is sufficient to renew the certificate alone, and not the document-certificate pair. This fails, of course, if the compromise in question is a method of computing hash collisions for the hash function used in submitting time-stamp requests.)

5 Different methods of time-stamping

To date, three different digital time-stamping techniques have been proposed: linear linking, random witness and linking into trees. What is the relationship between them? Does one supersede the others? Initially, one might think that trees satisfy time-stamping requirements better than the two previously proposed methods, because the tree protocol seems to reduce storage requirements while increasing the number of interested parties who serve as witnesses. But there are other tradeoffs to consider.

First we consider the linking protocol. In certain applications, such as a laboratory notebook, it is crucial not only to have a trustworthy date for each entry but also to establish in a trustworthy manner the exact sequence in which all entries were made. Linear linking of one entry to the next provides the most straightforward means of achieving this.

Next we consider the difference between the random-witness method and the tree method. While trees increase the number of witnesses to a given time-stamping event in proportion to the number of documents time-stamped, they do not guarantee a minimum number of witnesses. Neither do they guarantee that witnesses will retain their records. In contrast, in random witness the effective number of witnesses is the entire population, though only a small fraction are actually involved in any given time-stamping event. Furthermore, the set of signatures computed by the random-witness protocol explicitly creates a certificate which is evidence that a time-stamping event was widely witnessed. Thus, the protocol does not depend for its final valid-

ity on witnesses keeping records. Random witness is somewhat analogous to placing an advertisement in the newspaper, as discussed earlier, but with an additional refinement. Like the newspaper ad, it is effectively a widely witnessed event, but in addition it creates a record of the witnessing.

Given these tradeoffs, we imagine that the three methods may be used in a complementary fashion, as the following example illustrates. An individual or company might use linear linking to time-stamp its own accounting records, sending the final summary value for a given time period to a service maintained by a group of individuals or parties. This service constructs linked trees at regular intervals. The root of each tree is then certified as a widely viewed event by using the random-witness protocol among the participants. In this way, individual and group storage needs can be minimized, and the number of events which require an official record of witnessing can be greatly reduced.

References

[BC+ 88] B. O. Brachtl, D. Coppersmith, M. M. Hyden, S. M. Matyas, Jr., C. H. W. Meyer, J. Oseas, Sh. Pilpel, and M. Shilling. Data authentication using modification detection codes based on a public one way encryption function. U.S. Patent No. 4,908,861, issued March 13, 1990. (*Cf.* C. H. Meyer and M. Shilling, Secure program load with modification detection code. In *Securicom 88: 6ème Congrès mondial de la protection et de la sécurité informatique et des communications*, pp. 111–130 (Paris, 1988).)

[Dam 87] I. Damgård. Collision-free hash functions and public-key signature schemes. In *Advances in Cryptology—Eurocrypt '87*, Lecture Notes in Computer Science, Vol. 304, pp. 203–217, Springer-Verlag (Berlin, 1988).

[HaSt 91] S. Haber, W. S. Stornetta, How to time-stamp a digital document, *Journal of Cryptography*, Vol. 3, No. 2, pp. 99–111 (1991). (Presented at Crypto '90.)

[Merk 80] R. C. Merkle, Protocols for public key cryptosystems. In *Proc. 1980 Symp. on Security and Privacy*, IEEE Computer Society, pp. 122–133 (Apr. 1980).

[Riv 90] R. L. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology—Crypto '90*, Lecture Notes in Computer Science, Vol. 537 (ed. A. J. Menezes, S. A. Vanstone), pp. 303–311, Springer-Verlag (Berlin, 1991).

How to Time-Stamp a Digital Document*

Stuart Haber
stuart@bellcore.com

W. Scott Stornetta
stornetta@bellcore.com

Bellcore
115 South Street
Morristown, N.J. 07960-1910

Abstract

The prospect of a world in which all text, audio, picture, and video documents are in digital form on easily modifiable media raises the issue of how to certify when a document was created or last changed. The problem is to time-stamp the data, not the medium. We propose computationally practical procedures for digital time-stamping of such documents so that it is infeasible for a user either to back-date or to forward-date his document, even with the collusion of a time-stamping service. Our procedures maintain complete privacy of the documents themselves, and require no record-keeping by the time-stamping service.

*Appeared, with minor editorial changes, in *Journal of Cryptology*, Vol. 3, No. 2, pp. 99–111, 1991.

Time's glory is to calm contending kings,
To unmask falsehood, and bring truth to light,
To stamp the seal of time in aged things,
To wake the morn, and sentinel the night,
To wrong the wronger till he render right.

The Rape of Lucrece, l. 941

1 Introduction

In many situations there is a need to certify the date a document was created or last modified. For example, in intellectual property matters, it is sometimes crucial to verify the date an inventor first put in writing a patentable idea, in order to establish its precedence over competing claims.

One accepted procedure for time-stamping a scientific idea involves daily notations of one's work in a lab notebook. The dated entries are entered one after another in the notebook, with no pages left blank. The sequentially numbered, sewn-in pages of the notebook make it difficult to tamper with the record without leaving telltale signs. If the notebook is then stamped on a regular basis by a notary public or reviewed and signed by a company manager, the validity of the claim is further enhanced. If the precedence of the inventor's ideas is later challenged, both the physical evidence of the notebook and the established procedure serve to substantiate the inventor's claims of having had the ideas on or before a given date.

There are other methods of time-stamping. For example, one can mail a letter to oneself and leave it unopened. This ensures that the enclosed letter was created before the time postmarked on the envelope. Businesses incorporate more elaborate procedures into their regular order of business to enhance the credibility of their internal documents, should they be challenged at a later date. For example, these methods may ensure that the records are handled by more than one person, so that any tampering with a document by one person will be detected by another. But all these methods rest on two assumptions. First, the records can be examined for telltale signs of tampering. Second, there is another party that views the document whose integrity or impartiality is seen as vouchsafing the claim.

We believe these assumptions are called into serious question for the case of documents created and preserved exclusively in digital form. This is because electronic digital documents are so easy to tamper with, and the change needn't leave any telltale sign on the physical medium. What is needed is a method of time-stamping digital documents with the following two properties: First, one must find a way to time-stamp the data itself, without any reliance on the characteristics of the medium on which the data appears, so that it is impossible to change even one bit of the document without the change being apparent. Second, it should be impossible to stamp a document with a time and date different from the actual one.

The purpose of this paper is to introduce a mathematically sound and computationally practical solution to the time-stamping problem. In the sections that follow, we first consider a naive solution to the problem, the digital safety deposit box. This serves the pedagogical purpose of highlighting additional difficulties associated with digital time-stamping beyond those found in conventional methods of time-stamping. Successive improvements to this naive solution finally lead to practical

ways to implement digital time-stamping.

2 The Setting

The setting for our problem is a distributed network of users, perhaps representing individuals, different companies, or divisions within a company; we will refer to the users as *clients*. Each client has a unique identification number.

A solution to the time-stamping problem may have several parts. There is a procedure that is performed immediately when a client desires to have a document time-stamped. There should be a method for the client to verify that this procedure has been correctly performed. There should also be a procedure for meeting a third party's challenge to the validity of a document's time-stamp.

As with any cryptographic problem, it is a delicate matter to characterize precisely the security achieved by a time-stamping scheme. A good solution to the time-stamping problem is one for which, under reasonable assumptions about the computational abilities of the users of the scheme and about the complexity of a computational problem, and possibly about the trustworthiness of the users, it is difficult or impossible to produce false time-stamps. Naturally, the weaker the assumptions needed, the better.

3 A Naive Solution

A naive solution, a "digital safety-deposit box," could work as follows. Whenever a client has a document to be time-stamped, he or she transmits the document to a time-stamping service (TSS). The service records the date and time the document was received and retains a copy of the document for safe-keeping. If the integrity of the client's document is ever challenged, it can be compared to the copy stored by the TSS. If they are identical, this is evidence that the document has not been tampered with after the date contained in the TSS records. This procedure does in fact meet the central requirement for the time-stamping of a digital document.[1] However, this approach raises several concerns:

Privacy This method compromises the privacy of the document in two ways: a third party could eavesdrop while the document is being transmitted, and after transmission it is available indefinitely to the TSS itself. Thus the client has to worry not only about the security of documents it keeps under its direct control, but also about the security of its documents at the TSS.

Bandwidth and storage Both the amount of time required to send a document for time-stamping and the amount of storage required at the TSS depend on the length of the document to be time-stamped. Thus the time and expense required to time-stamp a large document might be prohibitive.

Incompetence The TSS copy of the document could be corrupted in transmission to the TSS, it could be incorrectly time-stamped when it arrives at the TSS, or it could become corrupted

[1] The authors recently learned of a similar proposal sketched by Kanare [14].

or lost altogether at any time while it is stored at the TSS. Any of these occurences would invalidate the client's time-stamping claim.

Trust The fundamental problem remains: nothing in this scheme prevents the TSS from colluding with a client in order to claim to have time-stamped a document for a date and time different from the actual one.

In the next section we describe a solution that addresses the first three concerns listed above. The final issue, trust, will be handled separately and at greater length in the following section.

4 A Trusted Time-Stamping Service

In this section we assume that the TSS is trusted, and describe two improvements on the naive solution above.

4.1 Hash

Our first simplification is to make use of a family of cryptographically secure *collision-free hash functions*. This is a family of functions $h : \{0,1\}^* \rightarrow \{0,1\}^l$ compressing bit-strings of arbitrary length to bit-strings of a fixed length l, with the following properties:

1. The functions h are easy to compute, and it is easy to pick a member of the family at random.
2. It is computationally infeasible, given one of these functions h, to find a pair of distinct strings x, x' satisfying $h(x) = h(x')$. (Such a pair is called a *collision* for h.)

The practical importance of such functions has been known for some time, and researchers have used them in a number of schemes; see, for example, [7, 15, 16]. Damgård gave the first formal definition, and a constructive proof of their existence, on the assumption that there exist one-way "claw-free" permutations [4]. For this, any "one-way group action" is sufficient [3].

Naor and Yung defined the similar notion of "universal one-way hash functions," which satisfy, in place of the second condition above, the slightly weaker requirement that it be computationally infeasible, given a string x, to compute another string $x' \neq x$ satisfying $h(x) = h(x')$ for a randomly chosen h. They were able to construct such functions on the assumption that there exist one-to-one one-way functions [17]. Rompel has recently shown that such functions exist if there exist one-way functions at all [20]. See §6.3 below for a discussion of the differences between these two sorts of cryptographic hash functions.

There are practical implementations of hash functions, for example that of Rivest [19], which seem to be reasonably secure.

We will use the hash functions as follows. Instead of transmitting his document x to the TSS, a client will send its hash value $h(x) = y$ instead. For the purposes of authentication, time-stamping y is equivalent to time-stamping x. This greatly reduces the bandwidth problem and the storage requirements, and solves the privacy issue as well. Depending on the design goals for an implementation of time-stamping, there may be a single hash function used by everybody, or different hash functions for different users.

For the rest of this paper, we will speak of time-stamping hash values y—random-appearing bit-strings of a fixed length. Part of the procedure for validating a time-stamp will be to produce the pre-image document x that satisfies $h(x) = y$; inability to produce such an x invalidates the putative time-stamp.

4.2 Signature

The second improvement makes use of digital signatures. Informally, a *signature scheme* is an algorithm for a party, the signer, to tag messages in a way that uniquely identifies the signer. Digital signatures were proposed by Rabin and by Diffie and Hellman [18, 7]. After a long sequence of papers by many authors, Rompel [20] showed that the existence of one-way functions can be used in order to design a signature scheme satisfying the very strong notion of security that was first defined by Goldwasser, Micali, and Rivest [10].

With a secure signature scheme available, when the TSS receives the hash value, it appends the date and time, then signs this compound document and sends it to the client. By checking the signature, the client is assured that the TSS actually did process the request, that the hash was correctly received, and that the correct time is included. This takes care of the problem of present and future incompetence on the part of the TSS, and completely eliminates the need for the TSS to store records.

5 Two Time-Stamping Schemes

Sed quis custodiet ipsos Custodes?
Juvenal, c. 100 A.D.
But who will guard the guards themselves?

What we have described so far is, we believe, a practical method for time-stamping digital documents of arbitrary length. However, neither the signature nor the use of hash functions in any way prevents a time-stamping service from issuing a false time-stamp. Ideally, we would like a mechanism which guarantees that no matter how unscrupulous the TSS is, the times it certifies will always be the correct ones, and that it will be unable to issue incorrect time-stamps even if it tries to.

It may seem difficult to specify a time-stamping procedure so as to make it impossible to produce fake time-stamps. After all, if the output of an algorithm A, given as input a document x and some timing information τ, is a bit-string $c = A(x, \tau)$ that stands as a legitimate time-stamp for x, what is to prevent a forger some time later from computing the same timing information τ and then running A to produce the same certificate c? The question is relevant even if A is a probabilistic algorithm.

Our task may be seen as the problem of simulating the action of a trusted TSS, in the absence of generally trusted parties. There are two rather different approaches we might take, and each one leads to a solution. The first approach is to constrain a centralized but possibly untrustworthy TSS to produce genuine time-stamps, in such a way that fake ones are difficult to produce. The second approach is somehow to distribute the required trust among the users of the service. It is not clear that either of these can be done at all.

5.1 Linking

Our first solution begins by observing that the sequence of clients requesting time-stamps and the hashes they submit cannot be known in advance. So if we include bits from the previous sequence of client requests in the signed certificate, then we know that the time-stamp occurred after these requests. But the requirement of including bits from previous documents in the certificate also can be used to solve the problem of constraining the time in the other direction, because the time-stamping company cannot issue later certificates unless it has the current request in hand.

We describe two variants of this linking scheme; the first one, slightly simpler, highlights our main idea, while the second one may be preferable in practice. In both variants, the TSS will make use of a collision-free hash function, to be denoted H. This is in addition to clients' use of hash functions in order to produce the hash value of any documents that they wish to have time-stamped.

To be specific, a time-stamping *request* consists of an l-bit string y (presumably the hash value of the document) and a client identification number ID. We use $\sigma(\cdot)$ to denote the signing procedure used by the TSS. The TSS issues signed, sequentially numbered time-stamp *certificates*. In response to the request (y_n, ID_n) from our client, the nth request in sequence, the TSS does two things:

1. The TSS sends our client the signed certificate $s = \sigma(C_n)$, where the certificate

$$C_n = (n, t_n, \text{ID}_n, y_n; L_n)$$

   consists of the sequence number n, the time t_n, the client number ID_n and the hash value y_n from the request, and certain *linking information*, which comes from the previously issued certificate: $L_n = (t_{n-1}, \text{ID}_{n-1}, y_{n-1}, H(L_{n-1}))$.

2. When the next request has been processed, the TSS sends our client the identification number ID_{n+1} for that next request.

Having received s and ID_{n+1} from the TSS, she checks that s is a valid signature of a good certificate, i.e. one that is of the correct form $(n, t, \text{ID}_n, y_n; L_n)$, containing the correct time t.

If her time-stamped document x is later challenged, the challenger first checks that the time-stamp (s, ID_{n+1}) is of the correct form (with s being a signature of a certificate that indeed contains a hash of x). In order to make sure that our client has not colluded with the TSS, the challenger can call client ID_{n+1} and ask him to produce his time-stamp (s', ID_{n+2}). This includes a signature

$$s' = \sigma(n+1, t_{n+1}, \text{ID}_{n+1}, y_{n+1}; L_{n+1})$$

of a certificate that contains in its linking information L_{n+1} a copy of her hash value y_n. This linking information is further authenticated by the inclusion of the image $H(L_n)$ of her linking information L_n. An especially suspicious challenger now can call up client ID_{n+2} and verify the next time-stamp in the sequence; this can continue for as long as the challenger wishes. Similarly, the challenger can also follow the chain of time-stamps backward, beginning with client ID_{n-1}.

Why does this constrain the TSS from producing bad time-stamps? First, observe that the use of the signature has the effect that the *only* way to fake a time-stamp is with the collaboration of the TSS. But the TSS cannot forward-date a document, because the certificate must contain bits from requests that immediately preceded the desired time, yet the TSS has not received them. The TSS

cannot feasibly back-date a document by preparing a fake time-stamp for an earlier time, because bits from the document in question must be embedded in certificates immediately following that earlier time, yet these certificates have already been issued. Furthermore, correctly embedding a new document into the already-existing stream of time-stamp certificates requires the computation of a collision for the hash function H.

Thus the only possible spoof is to prepare a fake chain of time-stamps, long enough to exhaust the most suspicious challenger that one anticipates.

In the scheme just outlined, clients must keep all their certificates. In order to relax this requirement, in the second variant of this scheme we link each request not just to the next request but to the next k requests. The TSS responds to the nth request as follows:

1. As above, the certificate C_n is of the form $C_n = (n, t_n, \mathrm{ID}_n, y_n; L_n)$, where now the linking information L_n is of the form

$$L_n = [(t_{n-k}, \mathrm{ID}_{n-k}, y_{n-k}, H(L_{n-k})), \ldots, (t_{n-1}, \mathrm{ID}_{n-1}, y_{n-1}, H(L_{n-1}))].$$

2. After the next k requests have been processed, the TSS sends our client the list $(\mathrm{ID}_{n+1}, \ldots, \mathrm{ID}_{n+k})$.

After checking that this client's time-stamp is of the correct form, a suspicious challenger can ask any one of the next k clients ID_{n+i} to produce his time-stamp. As above, his time-stamp includes a signature of a certificate that contains in its linking information L_{n+i} a copy of the relevant part of the challenged time-stamp certificate C_n, authenticated by the inclusion of the hash by H of the challenged client's linking information L_n. His time-stamp also includes client numbers $(\mathrm{ID}_{n+i+1}, \ldots, \mathrm{ID}_{n+i+k})$, of which the last i are new ones; the challenger can ask these clients for their time-stamps, and this can continue for as long as the challenger wishes.

In addition to easing the requirement that clients save all their certificates, this second variant also has the property that correctly embedding a new document into the already-existing stream of time-stamp certificates requires the computation of a simultaneously k-wise collision for the hash function H, instead of just a pairwise collision.

5.2 Distributed trust

For this scheme, we assume that there is a secure signature scheme so that each user can sign messages, and that a standard secure pseudorandom generator G is available to all users. A *pseudorandom generator* is an algorithm that stretches short input *seeds* to output sequences that are indistinguishable by any feasible algorithm from random sequences; in particular, they are unpredictable. Such generators were first studied by Blum and Micali [2] and by Yao [22]; Impagliazzo, Levin, and Luby have shown that they exist if there exist one-way functions [12].

Once again, we consider a hash value y that our client would like to time-stamp. She uses y as a seed for the pseudorandom generator, whose output can be interpreted in a standard way as a k-tuple of client identification numbers:

$$G(y) = (\mathrm{ID}_1, \mathrm{ID}_2, \ldots, \mathrm{ID}_k).$$

Our client sends her request (y, ID) to each of these clients. She receives in return from client ID_j a signed message $s_j = \sigma_j(t, \text{ID}, y)$ that includes the time t. Her time-stamp consists of $[(y, \text{ID}), (s_1, \ldots, s_k)]$. The k signatures s_j can easily be checked by our client or by a would-be challenger. No further communication is required in order to meet a later challenge.

Why should such a list of signatures constitute a believable time-stamp? The reason is that in these circumstances, the only way to produce a time-stamped document with an incorrect time is to use a hash value y so that $G(y)$ names k clients that are willing to cooperate in faking the time-stamp. If at any time there is at most a constant fraction ϵ of possibly dishonest clients, the expected number of seeds y that have to be tried before finding a k-tuple $G(y)$ containing only collaborators from among this fraction is ϵ^{-k}. Furthermore, since we have assumed that G is a secure pseudorandom generator, there is no faster way of finding such a convenient seed y than by choosing it at random. This ignores the adversary's further problem, in most real-world scenarios, of finding a plausible document that hashes to a convenient value y.

The parameter k should be chosen when designing the system so that this is an infeasible computation. Observe that even a highly pessimistic estimate of the percentage of the client population that is corruptible—ϵ could be 90%—does not entail a prohibitively large choice of k. In addition, the list of corruptible clients need not be fixed, as long their fraction of the population never exceeds ϵ.

This scheme need not use a centralized TSS at all. The only requirements are that it be possible to call up other clients at will and receive from them the required signatures, and that there be a public directory of clients so that it is possible to interpret the output of $G(y)$ in a standard way as a k-tuple of clients. A practical implementation of this method would require provisions in the protocol for clients that cannot be contacted at the time of the time-stamping request. For example, for suitable $k' < k$, the system might accept signed responses from any k' of the k clients named by $G(y)$ as a valid time-stamp for y (in which case a greater value for the parameter k would be needed in order to achieve the same low probability of finding a set of collaborators at random).

6 Remarks

6.1 Tradeoffs

There are a number of tradeoffs between the two schemes. The distributed-trust scheme has the advantage that all processing takes place when the request is made. In the linking scheme, on the other hand, the client has a short delay while she waits for the second part of her certificate; and meeting a later challenge may require further communication.

A related disadvantage of the linking scheme is that it depends on at least some clients storing their certificates.

The distributed-trust scheme makes a greater technological demand on the system: the ability to call up and demand a quick signed response at will.

The linking scheme only locates the time of a document between the times of the previous and the next requests, so it is best suited to a setting in which relatively many documents are submitted for time-stamping, compared to the scale at which the timing matters.

It is worth remarking that the time-constraining properties of the linking scheme do not depend

on the use of digital signatures.

6.2 Time constraints

We would like to point out that our schemes constrain the event of time-stamping both forward and backward in time. However, if any amount of time may pass between the creation of a document and when it is time-stamped, then no method can do more than forward-constrain the time at which the document itself was created. Thus, in general, time-stamping should only be considered as evidence that a document has not been back-dated.

On the other hand, if the time-stamping event can be made part of the document creation event, then the constraint holds in both directions. For example, consider the sequence of phone conversations that pass through a given switch. In order to process the next call on this switch, one could require that linking information be provided from the previous call. Similarly, at the end of the call, linking information would be passed onto the next call. In this way, the document creation event (the phone call) includes a time-stamping event, and so the time of the phone call can be fixed in both directions. The same idea could apply to sequential financial transactions, such as stock trades or currency exchanges, or any sequence of electronic interactions that take place over a given physical connection.

6.3 Theoretical considerations

Although we will not do it here, we suggest that a precise complexity-theoretic definition of the strongest possible level of time-stamping security could be given along the lines of the definitions given by Goldwasser and Micali [9], Goldwasser, Micali, and Rivest [10], and Galil, Haber, and Yung [8] for various cryptographic tasks. The time-stamping and the verification procedures would all depend on a *security parameter* p. A time-stamp scheme would be *polynomially secure* if the success probability of a polynomially bounded adversary who tries to manufacture a bogus time-stamp is smaller than any given polynomial in $1/p$ for sufficiently large p.

Under the assumption that there exist one-way claw-free permutations, we can prove our linking scheme to be polynomially secure. If we assume that there is always at most a constant fraction of corruptible clients, and assuming as well the existence of one-way functions (and therefore the existence of pseudorandom generators and of a secure signature scheme), we can prove our distributed-trust scheme to be polynomially secure.

In §4.1 above, we mentioned the difference between "collision-free" and "universal one-way" hash functions. The existence of one-way functions is sufficient to give us universal one-way hash functions. However, in order to prove the security of our time-stamping schemes, we apparently need the stronger guarantee of the difficulty of producing hash collisions that is provided by the definition of collision-free hash functions. As far as is currently known, a stronger complexity assumption—namely, the existence of claw-free pairs of permutations—is needed in order to prove the existence of these functions. (See also [5] and [6] for further discussion of the theoretical properties of cryptographic hash functions.)

Universal one-way hash functions were the tool used in order to construct a secure signature scheme. Our apparent need for a stronger assumption suggests a difference, perhaps an essential

one, between signatures and time-stamps. It is in the signer's own interest to act correctly in following the instructions of a secure signature scheme (for example, in choosing a hash function at random from a certain set). For time-stamping, on the other hand, a dishonest user or a colluding TSS may find it convenient not to follow the standard instructions (for example, by choosing a hash function so that collisions are easy to find); the time-stamping scheme must be devised so that there is nothing to be gained from such misbehavior.

If it is possible, we would like to reduce the assumptions we require for secure time-stamping to the simple assumption that one-way functions exist. This is the minimum reasonable assumption for us, since all of complexity-based cryptography requires the existence of one-way functions [12, 13]

6.4 Practical considerations

As we move from the realm of complexity theory to that of practical cryptosystems, new questions arise. In one sense, time-stamping places a heavier demand on presumably one-way functions than would some other applications. For example, if an electronic funds transfer system relies on a one-way function for authentication, and that function is broken, then all of the transfers carried out before it was broken are still valid. For time-stamps, however, if the hash function is broken, then all of the time-stamps issued prior to that time are called into question.

A partial answer to this problem is provided by the observation that time-stamps can be renewed. Suppose we have two time-stamping implementations, and that there is reason to believe that the first implementation will soon be broken. Then certificates issued using the old implementation can be renewed using the new implementation. Consider a time-stamp certificate created using the old implementation that is time-stamped with the new implementation before the old one is broken. Prior to the old implementation's breaking, the only way to create a certificate was by legitimate means. Thus, by time-stamping the certificate itself with the new implementation, one has evidence not only that the document existed prior to the time of the new time-stamp, but that it existed at the time stated in the original certificate.

Another issue to consider is that producing hash collisions alone is not sufficient to break the time-stamping scheme. Rather, meaningful documents must be found which lead to collisions. Thus, by specifying the format of a document class, one can complicate the task of finding meaningful collisions. For example, the density of ASCII-only texts among all possible bit-strings of length N bytes is $(2^7/2^8)^N$, or $1/2^N$, simply because the high-order bit of each byte is always 0. Even worse, the density of acceptable English text can be bounded above by an estimate of the entropy of English as judged by native speakers [21]. This value is approximately 1 bit per ASCII character, giving a density of $(2^1/2^8)^N$, or $1/128^N$.

We leave it to future work to determine whether one can formalize the increased difficulty of computing collisions if valid documents are sparsely and perhaps randomly distributed in the input space. Similarly, the fact that a k-way linking scheme requires the would-be adversary to compute k-way collisions rather than collision pairs may be parlayed into relaxing the requirements for the hash function. It may also be worthwhile to explore when there exist hash functions for which there are *no* k-way collisions among strings in a suitably restricted subset of the input space; the security of such a system would no longer depend on a complexity assumption.

7 Applications

Using the theoretically best (cryptographically secure) hash functions, signature schemes, and pseudorandom generators, we have designed time-stamping schemes that possess theoretically desirable properties. However, we would like to emphasize the practical nature of our suggestion: because there are *practical* implementations of these cryptographic tools, both of our time-stamp schemes can be inexpensively implemented as described. Practical hash functions like Rivest's are quite fast, even running on low-end PC's [19].

What kinds of documents would benefit from secure digital time-stamping? For documents that establish the precedence of an invention or idea, time-stamping has a clear value. A particularly desirable feature of digital time-stamping is that it makes it possible to establish precedence of intellectual property without disclosing its contents. This could have a significant effect on copyright and patent law, and could be applied to everything from software to the secret formula for Coca-Cola.

But what about documents where the date is not as significant as simply whether or not the document has been tampered with? These documents can benefit from time-stamping, too, under the following circumstances. Suppose one can establish that either the necessary knowledge or the motivation to tamper with a document did not exist until long after the document's creation. For example, one can imagine a company that deals with large numbers of documents each day, some few of which are later found to be incriminating. If all the company's documents were routinely time-stamped at the time of their creation, then by the time it became apparent which documents were incriminating and how they needed to be modified, it would be too late to tamper with them. We will call such documents *tamper-unpredictable*. It seems clear that many business documents are tamper-unpredictable. Thus, if time-stamping were to be incorporated into the established order of business, the credibility of many documents could be enhanced.

A variation that may be particularly useful for business documents is to time-stamp a log of documents rather than each document individually. For example, each corporate document created in a day could be hashed, and the hash value added to the company's daily log of documents. Then, at the end of the business day, the log alone could be submitted for time-stamping. This would eliminate the expense of time-stamping each document individually, while still making it possible to detect tampering with each document; one could also determine whether any documents had been destroyed altogether.

Of course, digital time-stamping is not limited to text documents. Any string of bits can be time-stamped, including digital audio recordings, photographs, and full-motion videos. Most of these documents are tamper-unpredictable. Therefore, time-stamping can help to distinguish an original photograph from a retouched one, a problem that has received considerable attention of late in the popular press [1, 11]. It is in fact difficult to think of any other algorithmic "fix" that could add more credibility to photographs, videos, or audio recordings than time-stamping.

8 Summary

In this paper, we have shown that the growing use of text, audio and video documents in digital form and the ease with which such documents can be modified creates a new problem: how can one

certify when a document was created or last modified? Methods of certification, or time-stamping, must satisfy two criteria. First, they must time-stamp the actual bits of the document, making no assumptions about the physical medium on which the document is recorded. Second, the date and time of the time-stamp must not be forgeable.

We have proposed two solutions to this problem. Both involve the use of one-way hash functions, whose outputs are processed in lieu of the actual documents, and of digital signatures. The solutions differ only in the way that the date and time are made unforgeable. In the first, the hashes of documents submitted to a TSS are linked together, and certificates recording the linking of a given document are distributed to other clients both upstream and downstream from that document. In the second solution, several members of the client pool must time-stamp the hash. The members are chosen by means of a pseudorandom generator that uses the hash of the document itself as seed. This makes it infeasible to deliberately choose which clients should and should not time-stamp a given hash. The second method could be implemented without the need for a centralized TSS at all.

Finally, we have considered whether time-stamping could be extended to enhance the authenticity of documents for which the time of creation itself is not the critical issue. This is the case for a large class of documents which we call "tamper-unpredictable." We further conjecture that no purely algorithmic scheme can add any more credibility to a document than time-stamping provides.

Acknowledgements

We gratefully acknowledge helpful discussions with Donald Beaver, Shimon Even, George Furnas, Burt Kaliski, Ralph Merkle, Jeff Shrager, Peter Winkler, Yacov Yacobi, and Moti Yung.

References

[1] J. Alter. When photographs lie. *Newsweek*, pp. 44-45, July 30, 1990.

[2] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, Nov. 1984.

[3] G. Brassard and M. Yung. One-way group actions. In *Advances in Cryptology—Crypto '90*. Springer-Verlag, *LNCS*, to appear.

[4] I. Damgård. Collision-free hash functions and public-key signature schemes. In *Advances in Cryptology—Eurocrypt '87*, pp. 203-217. Springer-Verlag, *LNCS*, vol. 304, 1988.

[5] I. Damgård. A design principle for hash functions. In *Advances in Cryptology—Crypto '89* (ed. G. Brassard), pp. 416-427. Springer-Verlag, *LNCS*, vol. 435, 1990.

[6] A. DeSantis and M. Yung. On the design of provably secure cryptographic hash functions. In *Advances in Cryptology—Eurocrypt '90*. Springer-Verlag, *LNCS*, to appear.

[7] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Trans. on Inform. Theory*, vol. IT-22, Nov. 1976, pp. 644-654.

[8] Z. Galil, S. Haber, and M. Yung. Interactive public-key cryptosystems. *J. of Cryptology*, to appear.

[9] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28:270–299, April 1984.

[10] S. Goldwasser, S. Micali, and R. Rivest. A secure digital signature scheme. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[11] Andy Grundberg. Ask it no questions: The camera can lie. *The New York Times*, section 2, pp. 1, 29, August 12, 1990.

[12] R. Impagliazzo, L. Levin, and M. Luby. Pseudorandom generation from one-way functions. In *Proc. 21st STOC*, pp. 12-24. ACM, 1989.

[13] R. Impagliazzo and M. Luby. One-way functions are essential for complexity-based cryptography. In *Proc. 30th FOCS*, pp. 230-235. IEEE, 1989.

[14] H. M. Kanare. *Writing the laboratory notebook*, p. 117. American Chemical Society, 1985.

[15] R.C. Merkle. Secrecy, authentication, and public-key systems. Ph.D. thesis, Stanford Univeristy, 1979.

[16] R.C. Merkle. One-way hash functions and DES. In *Advances in Cryptology—Crypto '89* (ed. G. Brassard), pp. 428-446. Springer-Verlag, *LNCS*, vol. 435, 1990.

[17] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 21st STOC*, pp. 33-43. ACM, 1989.

[18] M.O. Rabin. Digitalized signatures. In *Foundations of Secure Computation* (ed. R.A. DeMillo et al.), pp. 155-168. Academic Press, 1978.

[19] R. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology—Crypto '90.* Springer-Verlag, *LNCS*, to appear.

[20] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. 22nd STOC*, pp. 387-394. ACM, 1990.

[21] C. Shannon. Prediction and entropy of printed English. *Bell System Technical Journal*, vol. 30 pp. 50-64, 1951.

[22] A.C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pp. 80-91. IEEE, 1982.